12. Functions

Pandas provides a wide range of mathematical functions for performing operations on data frames and series.
Here are some commonly used mathematical functions provided by pandas,

Aggregating functions,

1. **sum()** : Computes the sum of values

```
df['A'].sum()
```

2. **mean()** : Computes the mean of values

```
df['A'].mean()
```

3. **median()** : Computes the median of values

```
df['A'].median()
```

4. **std()** : Computes the standard deviation of values

```
df['A'].std()
```

5. **var()** : Computes the variance of values

```
df['A'].var()
```

6. **min()** : Computes the minimum of values

```
df['A'].min()
```

7. max()** : Computes the maximum of values

```
df['A'].max()
```

8. **count()** : Counts the number of non-null values.

```
df['A'].count()
```

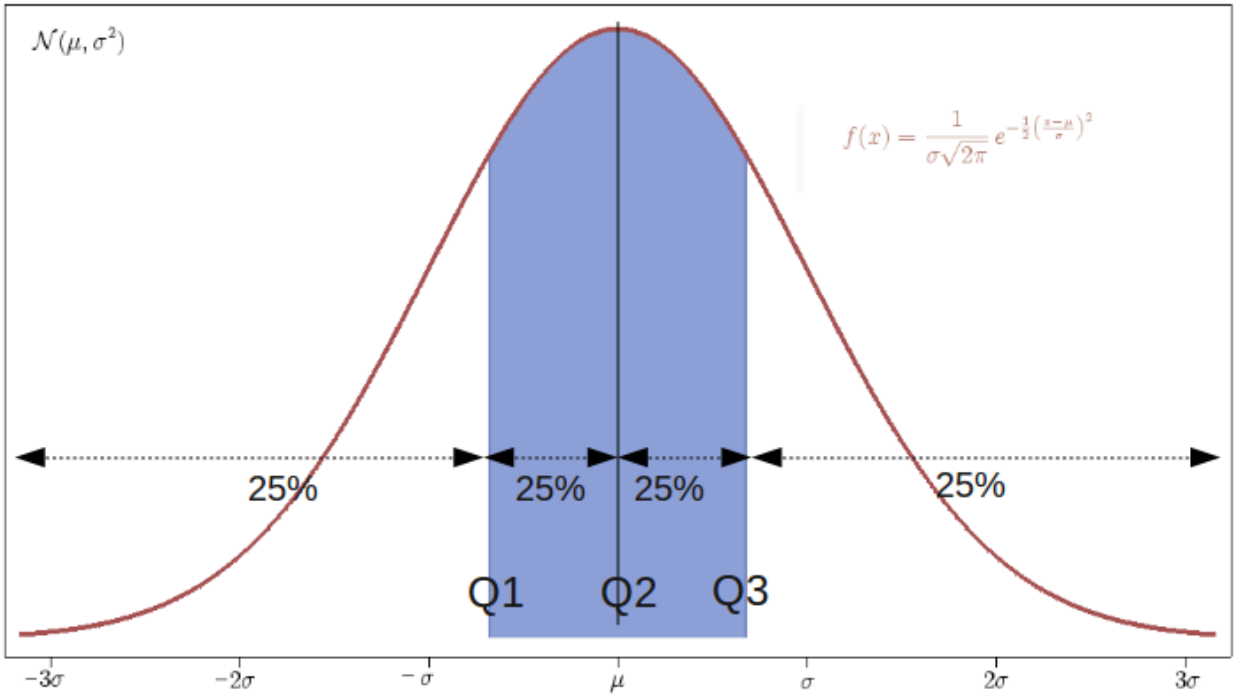
9. **mode()** : Computes the mode(most frequent value)

```
df['A'].mode()
```

10. **quantiles()** : Computes the quantile of values

```
df['A'].quantile(0.5)
```

In statistics and probability, quantiles are **cut points dividing the range of a probability distribution into continuous intervals with equal probabilities**, or dividing the observations in a sample in the same way.



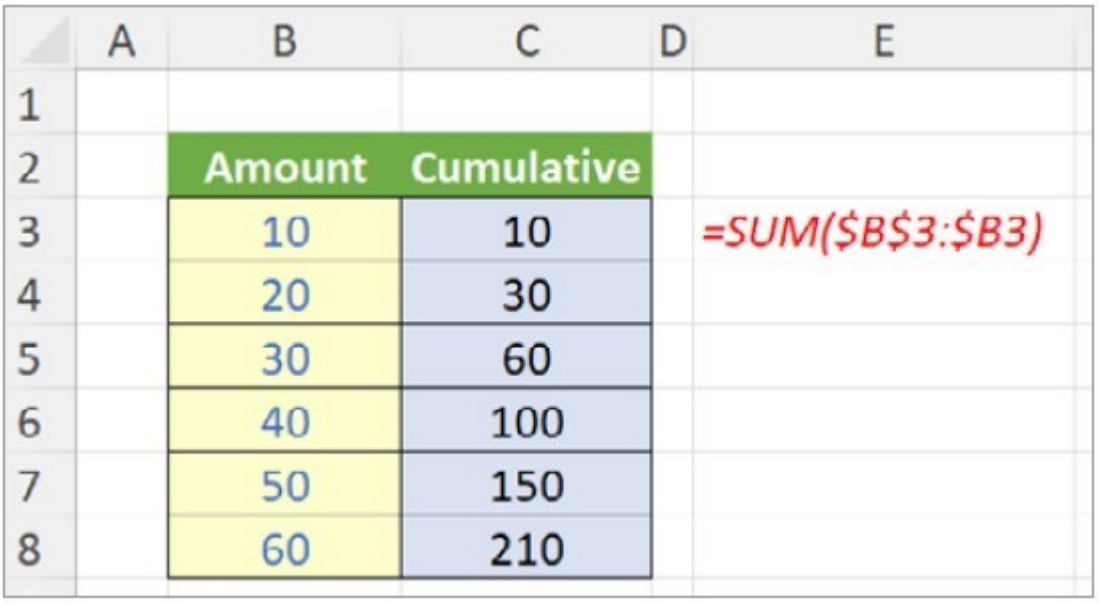
Cumulative Functions

1. **cumsum()** : Computes the cumulative sum

```
df['A'].cumsum()
```

A cumulative sum, also known as a **running total**, is the total of a series of data as it increases over time. It's calculated by adding the current value to the sum of all previous

values.



	A	B	C	D	E
1					
2		Amount	Cumulative		
3		10	10		=SUM(B3:$B3)
4		20	30		
5		30	60		
6		40	100		
7		50	150		
8		60	210		

2. **cumprod()** : Computes the cumulative product

```
df['A'].cumprod()
```

The cumulative product of a sequence is **a running products or partial products of a sequence**.
The cumulative products of the sequence `{a,b,c,…}` are
a, a*b*, *a*b*c, ….

```
Vector :
     3     5   NaN     9     0   NaN

Cummulative product Incude NaN :
     3    15   NaN   NaN   NaN   NaN

Cummulative product Exclude NaN :
     3    15    15   135     0     0
```

3. **cummin()** : Computes the cumulative minimum

```
df['A'].cummin()
```

The cumulative minimum identifies the minimum value of a column up to and including the current row value.

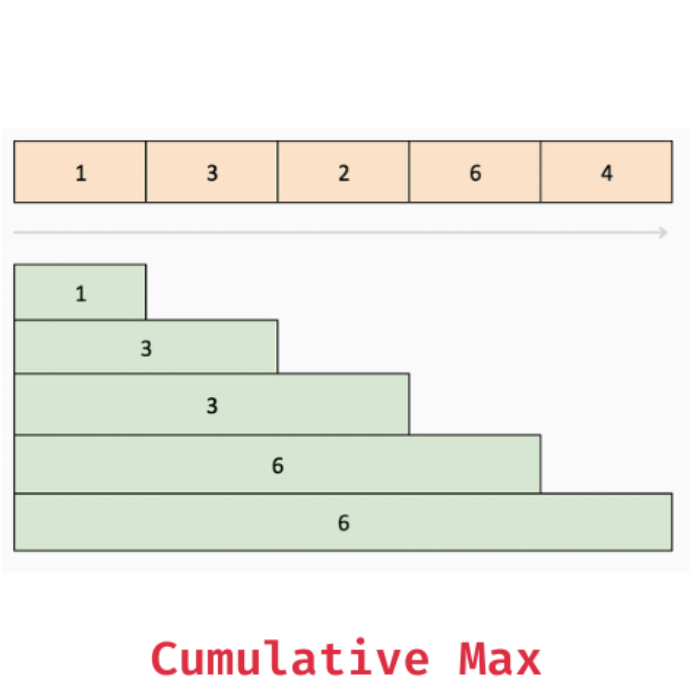
- **In a DataFrame**: The cumulative minimum is the lowest value for each column in a DataFrame

Date	Sales	CumulativeMin of Sales
Jan 2024	100,000	100,000
Feb 2024	95,000	95,000
Mar 2024	110,000	95,000
Apr 2024	80,000	80,000
May 2024	90,000	80,000
Jun 2024	75,000	75,000
Jul 2024	100,000	75,000
Aug 2024	105,000	75,000
Sep 2024	95,000	75,000
Oct 2024	120,000	75,000
Nov 2024	125,000	75,000
Dec 2024	120,000	75,000

4. **cummax()** : Computes the cumulative maximum

```
df['A'].cummax()
```

Cumulative maximum refers to the maximum value of a column, including the current row value.



Descriptive Statistics

1. **describe()** : Generates descriptive statistics

```
df.describe()
```

If the DataFrame contains numerical data, the description contains these information for each column: count - The number of not-empty values. mean - The average (mean) value. std - The standard deviation.

```
import pandas as pd

city_names = pd.Series(['San Francisco', 'San Jose', 'Sacramento'])
population = pd.Series([852469, 1015785, 485199])
cities = pd.DataFrame({ 'City name': city_names, 'Population': population })
cities['Area square miles'] = pd.Series([46.87, 176.53, 97.92])
cities['Population density'] = cities['Population'] / cities['Area square miles']
cities['Is wide and has saint name'] = (cities['Area square miles'] > 50) & cities['City name']\
    .apply(lambda name: name.startswith('San'))

cities.describe()
```

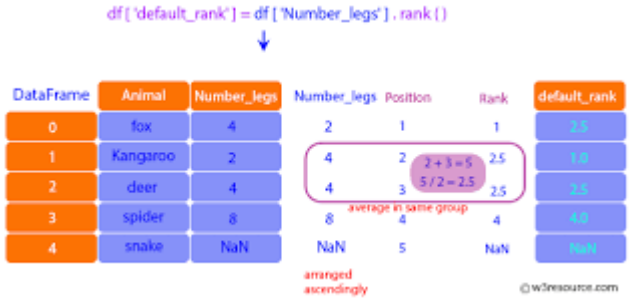
	Population	Area square miles	Population density
count	3.000000e+00	3.000000	3.000000
mean	7.844843e+05	107.106667	9632.392763
std	2.717477e+05	65.316346	7420.091623
min	4.851990e+05	46.870000	4955.055147
25%	6.688340e+05	72.395000	5354.616454
50%	8.524690e+05	97.920000	5754.177760
75%	9.341270e+05	137.225000	11971.061570
max	1.015785e+06	176.530000	18187.945381

Ranking Functions

1. **rank()** : Computes numerical data ranks

```
df['A'].rank()
```

```python
# Understanding the Pandas .rank() Method
import pandas as pd
df.rank(
    axis = 0,               # Rank columns/rows
    method = 'average',     # How to rank duplicate values
    numeric_only = True,    # Rank only numeric columns
    na_option = 'keep',     # How to rank missing data
    ascending = True,       # What order to rank in
    pct = False             # Rank in a normalized way or not
)
```



Difference Functions

1. **diff()** : Computes the difference of a DataFrame element compared with another element in the DataFrame(Default is element in the previous row)

```python
df['A'].diff()
```

main.py

```python
# Example: diff
import pandas as pd

# Create a dataframe
df = pd.DataFrame({'A': [1, 2, 3, 4, 5, 6],
                   'B': [1, 1, 2, 3, 5, 8],
                     'C': [1, 4, 9, 16, 25, 36]})

# Syntax: DataFrame.diff(periods=1, axis=0)

# Calculate the difference between the current and a shifted row

print(df.diff())
```

```
>>>      A    B     C
    0  NaN  NaN   NaN
    1  1.0  0.0   3.0
    2  1.0  1.0   5.0
    3  1.0  1.0   7.0
    4  1.0  2.0   9.0
    5  1.0  3.0  11.0
```

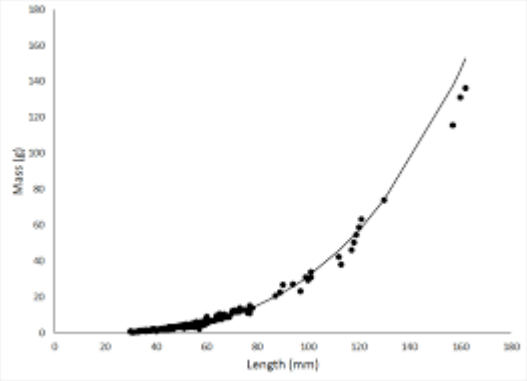
Expanding Transformation

1. **expanding()** : Provides expanding transformations.

```
df['A'].expanding().sum()
```

In statistics, an "expanding transformation" generally refers to a data transformation technique where the original data points are replaced with a new set of values that are progressively "stretched out" or expanded across a wider range, often achieved by applying a mathematical function that accentuates differences between data points, typically used to

improve the visibility of patterns or relationships in highly concentrated data sets.

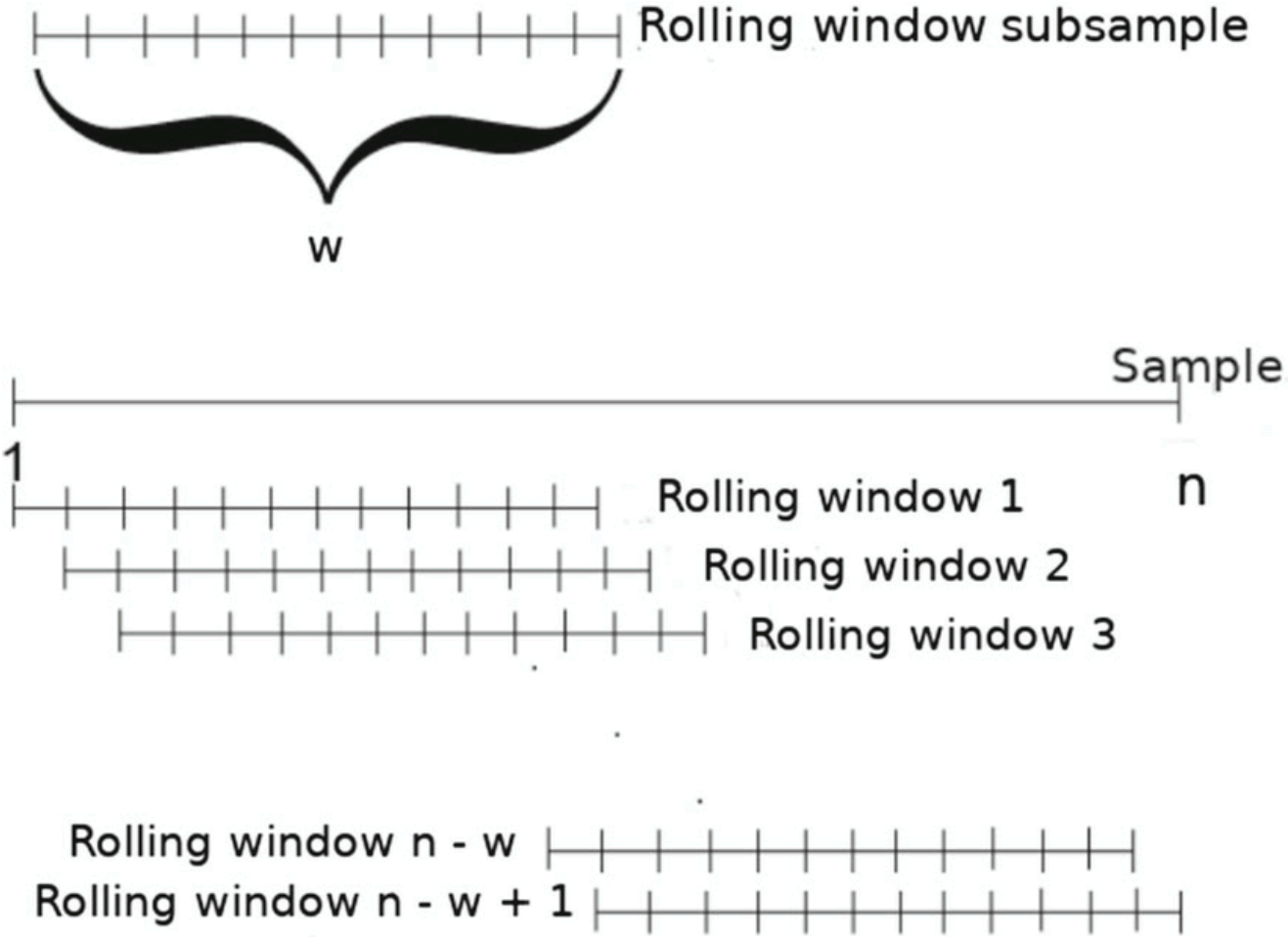


Rolling window functions

1. **rolling()** : Provides rolling window calculations.

```
df['A'].rolling(window=3).mean()
```

Rolling window calculations are a way to calculate statistics that move with data, such as averages, sums, or maximums.



Other mathematical functions

1. **abs()** : Computes the absolute values elementwise

```
df['A'].abs()
```

2. **round()** : Rounds values to a specified number of decimal places

```
df['A'].round(2)
```

3. **pow()** : Computes the power of elements

```
df['A'].pow(2)
```

3. **pct_change()** : Computes the percentage change between the current and a prior element

```
df['A'].pct_change()
```

3. **clip()** : Trims values at input threshold(s)

```
df['A'].clip(lower=0, upper=10)
```